Software Prototyping

The benefits of developing a prototype early in the software process are:

1. Misunderstandings between software developers and users may be identified as the system functions are demonstrated.
2. Missing user services may be detected.
3. Difficult-to-use or confusing user services may be identified and refined.
4. Software development staff may find incomplete and/or inconsistent requirements as the prototype is developed.
5. A working, albeit limited, system is available quickly to demonstrate the feasibility and usefulness of the application to management.
6. The prototype serves as a basis for writing the specification for a production quality system.

Software prototypes also have other uses:

1. User training
2. System testing

The final stage of the process is prototype evaluation.

There are non-technical managerial problems which may make it difficult to use prototyping in some organizations.

Prototyping in the software process

One way of tackling this difficulty is to use an evolutionary approach to systems development.

Alternatively, a deliberate decision might be made to build a 'throw-away' prototype to help requirements analysis and validation.

There is an important difference between the objectives of evolutionary and throw-away programming:
(1) The objective of evolutionary prototyping is to deliver a working system to end-users.

(2) The objective of throw-away prototyping is to validate or derive the system requirements.

**Evolutionary prototyping**
- There are three main problems with evolutionary prototyping which are particularly important when large, long-lifetime systems are to be developed:
  - Difficulties do not mean that evolutionary prototyping should be rejected.

**Throw-away prototyping**
- This approach extends the requirements analysis process with the intention of reducing overall life cycle costs.
  - Customers and end-users should resist the temptation to turn the throw-away prototyping into a delivered system that is put into use.

**Incremental development**
- This incremental development model involves developing the requirements and delivering the system in an incremental fashion.
  - Incremental development is more manageable than evolutionary prototyping as the normal software process standards are followed.
    - A problem with incremental development is that the system architecture has to be established before the requirements are complete.

**Prototyping techniques**
- There are a number of techniques which have been used for system prototyping. These include:
  + Executable specification languages,
  + Very high level language,
Software Prototyping

+ Application generators and fourth-generation languages,
+ Composition of reusable components.

**Executable specification languages**

- There are practical difficulties in applying this approach.

**Very high level languages**

- Very high level languages are programming languages which include powerful data management facilities.
- Very high level dynamic languages are not normally used for large system development because they need a large run-time support system.

**Fourth-generation languages**

- Fourth-generation languages are successful because there is a great deal of commonality across data processing applications.
- There is no standardization or uniformity across fourth-generation languages.

**Composition of reusable components**

- Prototyping with reusable components involves developing a system specification by taking account of what reusable components are available.
- Prototyping using reusable components is often combined with other approaches using very high level or fourth-generation languages.

**User interface prototyping**

- Interface generation systems may be based around user interface management systems which provide basic user interface functionality such as menu selection, object display and so on.
- From a software engineering point of view, it is important to realize that user interface prototyping is an essential part of the process.